(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 762 270 A2

(12) EUROPEAN PATENT APPLICATION

(43) Date of publication:
12.03.1997 Bulletin 1997/11

(51) Int Cl.6: G06F 9/312

(21) Application number: 96306249.2

(22) Date of filing: 29.08.1996

(84) Designated Contracting States:
DE FR GB

(30) Priority: 11.09.1995 US 526343

(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION
Armonk, NY 10504 (US)

(72) Inventors:
- Kahle, James A.
  Austin, Texas 78731 (US)
- Loper, Albert J.
  Cedar Park, Texas 78613 (US)
- Mallick, Soummya
  Austin, Texas 78729 (US)
- Ogden, Aubrey D.
  Round Rock, Texas 78681 (US)

(74) Representative: Davies, Simon Robert
I B M
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) Microprocessor with load/store operation to/from multiple registers

(57) A load multiple instruction may be executed in a superscaler microprocessor by dispatching a load multiple instruction to a load/store unit, wherein the load/store unit begins execution of a dispatched load multiple instruction, and wherein the load multiple instruction loads data from memory into a plurality of registers. A table is maintained that lists each register of the plurality of registers and that indicates when data has been loaded into each register by the executing load multiple instruction. An instruction is executed that is dependent upon source operand data loaded by the load multiple instruction, prior to the load multiple instruction completing its execution, when the table indicates the source operand data has been loaded into the source register. Also, a store multiple instruction may be executed by dispatching a store multiple instruction to the load/store unit, whereupon the load/store unit begins executing the store multiple instruction, wherein the load/store instruction stores data from a plurality of registers to memory. A fixed point instruction is executed that is dependent upon data being stored by the store multiple instruction prior to the store multiple instruction completing its execution, but the executing fixed point instruction is prohibited from writing to a register of the plurality of registers prior to the store multiple instruction completing.

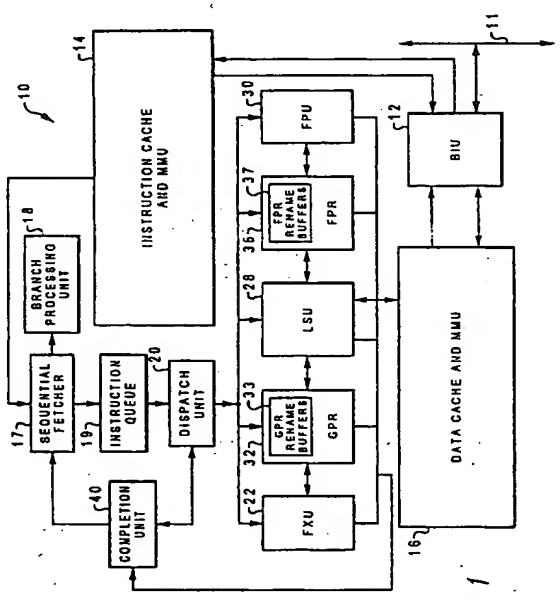

Fig. 1

EP 0 762 270 A2

Description

The present invention relates in general to the allocation of resources during the execution of instructions in a microprocessor, and in particular to the allocation of resources in microprocessors which have a load/store operation to multiple registers.

Multi-register load/store instructions require complete serialization since these instructions modify or use up to all the general purpose registers (usually 32) contained within a microprocessor. In the PowerPC™ line of microprocessors produced by International Business Machines Corporation, integer load/store multiple instructions are accommodated, which move blocks of data to and from the microprocessor's general purpose registers (GPRs). The multi-register instructions provided are the Load Multiple Word (lmw) and the Store Multiple Word (stmw) instructions.

In the prior art, because such multi-register load/store instructions may modify or use up to all the general purpose registers in the system, later instructions in the instruction sequence are held in the instruction buffer until the multi-register instruction is complete. Therefore, it was assumed that multi-register instructions force complete serialization of these instructions in the instruction stream. To implement such a serialization in the prior art, the multi-register instructions are allocated all of the required general purpose registers until the instruction is completed. Such a system substantially restricts performance by holding up the instruction pipeline until the multi-register instruction is completed.

Accordingly, the invention provides a method of executing multiple instructions in a superscalar microprocessor, including at least one load multiple instruction that loads to more than one register of a plurality of registers, comprising the steps of:

dispatching a load multiple instruction to a load/store unit, wherein the load/store unit begins execution of a dispatched load multiple instruction, and wherein the load multiple instruction loads data from memory into a plurality of registers;
maintaining a table that lists each register of the plurality of registers and that indicates when data has been loaded into each register by the executing load multiple instruction; and
executing an instruction that is dependent upon source operand data loaded by the load multiple instruction into a register of the plurality of registers indicated by the instruction as a source register, prior to the load multiple instruction completing its execution, when the table indicates the source operand data has been loaded into the source register.

The invention further provides a method of executing multiple instructions in a superscalar microprocessor, including at least one store multiple instruction that stores data from more than one register of a plurality of registers to memory, comprising the steps of:

dispatching a store multiple instruction to a load/store unit, whereupon the load/store unit begins executing the store multiple instruction, wherein the store multiple instruction stores data from a plurality of registers to memory; and
executing a fixed point instruction that is dependent upon data being stored by the store multiple instruction from a register of the plurality of registers indicated by the fixed point instruction as a source register, prior to the store multiple instruction completing its execution, but prohibiting the executing fixed point instruction from writing to a register of the plurality of registers prior to the store multiple instruction completing.

In the preferred embodiment, said dependent instruction does not complete until after said load or store multiple instruction has completed.

The invention further provides a superscalar microprocessor having a plurality of execution units for simultaneously executing a plurality of instructions and being connected to a memory, comprising:

a plurality of registers for selectively storing data resulting from execution by the execution units;
a load/store execution unit that executes load and store instructions which load or store data to or from the plurality of registers;
a fixed point execution unit for performing fixed point operations on operand data stored in source registers of the plurality of registers;
and a dispatcher that maintains a table listing at least some of the plurality of registers and indicating when data has been loaded into each register by a load multiple instruction executing in the load/store execution unit, wherein a load multiple instruction loads data from memory into more than one of the plurality of registers, and wherein the dispatcher dispatches instructions to the plurality of execution units, including the load/store execution unit and the fixed point execution unit, and further wherein the dispatcher dispatches a load multiple instruction to the load/store unit, which begins execution of the load multiple instruction and further, prior to the load multiple instruction finishing its execution, dispatches a fixed point instruction which is dependent upon source operand data loaded by the load multiple instruction into a register of the plurality of registers indicated by the instruction as a source register, when the table indicates the source operand data has been loaded into the source register by the execution of the load multiple instruction in the load/store execution unit.

The invention further provides a superscalar microprocessor having a plurality of execution units for simul-

taneously executing a plurality of instructions and being connected to a memory, comprising:

a plurality of registers for selectively storing data resulting from execution by the execution units;
a load/store execution unit that executes load and store instructions which load or store data to or from the plurality of registers;
a fixed point execution unit for performing fixed point operations on operand data stored in source registers of the plurality of registers;
a dispatcher that dispatches instructions to the plurality of execution units, including the load/store execution unit and the fixed point execution unit, wherein the dispatcher dispatches a store multiple instruction to the load/store unit, which begins execution of the store multiple instruction, wherein a store multiple instruction stores data from more than one of the plurality of registers to memory, and further, prior to the store multiple instruction finishing, dispatches a fixed point instruction, which is dependent upon source operand data stored in a register of the plurality of registers that is being stored out to memory by the executing store multiple instruction, to the fixed point execution unit, wherein the fixed point execution unit executes the dispatched fixed point instruction prior to the store multiple instruction finishing, but does not store the result of the executed fixed point instruction to a register of the plurality of registers prior to the store multiple instruction finishing.

In a preferred embodiment, the dispatcher in the microprocessor is capable of handling both load and store operations as described above.

The approach described above allows resources to be deallocated from the multi-register instructions as those instructions are executed so that the resources become available to allow subsequent instructions to begin execution simultaneously with the executing multi-register instruction. Thus the early deallocation of resources dedicated to a serialized load/store multiple operation allows simultaneous execution of additional instructions. This substantially improves the performance of a microprocessor having a superscalar design by allowing instructions that utilize multiple registers to be executed in parallel with subsequent instructions that utilize the general purpose registers.

Thus in a preferred embodiment, a load multiple instruction may be executed in a superscaler microprocessor by dispatching the load multiple instruction to a load/store unit. The load/store unit begins execution of a dispatched load multiple instruction, and the load multiple instruction loads data from memory into a plurality of registers. A table is maintained that lists each register of the plurality of registers and indicates when data has been loaded into each register by the executing load multiple instruction. An instruction that is dependent upon source operand data loaded by the load multiple instruction into a register of the plurality of registers indicated by the instruction as a source register is executed prior to the load multiple instruction completing its execution, after the table indicates the source operand data has been loaded into the source register.

Also in the preferred embodiment, a store multiple instruction may be executed in a superscaler microprocessor by dispatching the store multiple instruction to a load/store unit. The load/store unit begins executing the store multiple instruction, wherein the load store instruction stores data from a plurality of registers to memory. A fixed point instruction that is dependent upon data being stored by the store multiple instruction from a register of the plurality of registers indicated by the fixed point instruction as a source register is executed prior to the store multiple instruction completing its execution. However, the executing fixed point instruction is prohibited from writing to a register of the plurality of registers prior to the store multiple instruction completing.

A preferred embodiment of the invention will now be described by way of example only with reference to the following drawings:

Figure 1 illustrates a block diagram of a processor;
Figure 2 shows a timing diagram of the cycles required to handle a load multiple instruction and subsequent fixed-point instructions;
Figure 3 shows a timing diagram showing the cycles in which a store multiple instruction and two fixed-point instructions are handled in the microprocessor.

With reference now to the figures and in particular with reference to **Figure 1**, there is illustrated a block diagram of a processor, indicated generally at **10**, for processing information. In the depicted embodiment, processor **10** comprises a single integrated circuit superscalar microprocessor. Accordingly, as discussed further below, processor **10** includes various execution units, registers, buffers, memories, and other functional units, which are all formed by integrated circuitry. Processor **10** may comprise one of the PowerPC™ line of microprocessors produced by International Business Machines Corporation which operates according to reduced instruction set computing (RISC) techniques.

As depicted in **Figure 1**, processor **10** is coupled to system bus **11** via a bus interface unit (BIU) **12** within processor **10**. BIU **12** controls the transfer of information between processor **10** and other devices coupled to system bus **11**, such as a main memory (not illustrated). Processor **10**, system bus **11**, and the other devices coupled to system bus **11** together form a host data processing system. BIU **12** is connected to instruction cache **14** and data cache **16** within processor **10**. High speed caches, such as instruction cache **14** and data cache **16**, enable processor **10** to achieve relatively fast access time to a subset of data or instructions previously

transferred from main memory to the high speed caches, thus improving the speed of operation of the host data processing system. Instruction cache **14** is further coupled to sequential fetcher **17**, which fetches instructions from instruction cache **14** during each cycle for execution. Sequential fetcher **17** transfers branch instructions to branch processing unit (BPU) **18** for execution, and transfers sequential instructions to instruction queue **19** for temporary storage before being executed by other execution circuitry within processor **10**.

In the depicted embodiment, in addition to BPU **18**, the execution circuitry of processor **10** comprises multiple execution units, including fixed-point unit (FXU) **22**, load/store unit (LSU) **28**, and floating-point unit (FPU) **30**. As is well-known to those skilled in the computer art, each of execution units **22**, **28**, and **30** executes one or more instructions within a particular class of sequential instructions during each processor cycle. For example, FXU **22** performs fixed-point mathematical operations such as addition, subtraction, ANDing, ORing, and XORing, utilizing source operands received from specified general purpose registers (GPRs) **32** or GPR rename buffers **33**. Following the execution of a fixed-point instruction, FXU **22** outputs the data results of the instruction to GPR rename buffers **33**, which provide temporary storage for the result data until the instruction is completed by transferring the result data from GPR rename buffers **33** to one or more of GPRs **32**. Conversely, FPU **30** performs floating-point operations, such as floating-point multiplication and division, on source operands received from floating-point registers (FPRs) **36** or FPR rename buffers **37**. FPU **30** outputs data resulting from the execution of floating-point instructions to selected FPR rename buffers **37**, which temporarily store the result data until the instructions are completed by transferring the result data from FPR rename buffers **37** to selected FPRs **36**. LSU **28** executes floating-point and fixed-point instructions that either load data from memory (i.e., either data cache **16** or main memory) into selected GPRs **32** or FPRs **36**, or that store data from a selected one of GPRs **32**, GPR rename buffers **33**, FPRs **36**, or FPR rename buffers **37** to memory.

Processor **10** employs both pipelining and out-of-order execution of instructions to further improve the performance of its superscalar architecture. Accordingly, instructions can be executed by FXU **22**, LSU **28**, and FPU **30** in any order as long as data dependencies are observed. In addition, instructions are processed by each of FXU **22**, LSU **28**, and FPU **30** at a sequence of pipeline stages. As is typical of high-performance processors, each instruction is processed at five distinct pipeline stages, namely, fetch, decode/dispatch, execute, finish, and completion.

During the fetch stage, sequential fetcher **17** retrieves one or more instructions associated with one or more memory addresses from instruction cache **14**. Sequential instructions fetched from instruction cache **14** are stored by sequential fetcher **17** within instruction queue **19**. Fetched branch instructions are removed from the instruction stream and are forwarded to BPU **18** for execution. BPU **18** includes a branch prediction mechanism, such as a branch history table, that enables BPU **18** to speculatively execute unresolved conditional branch instructions by predicting whether the branch will be taken.

During the decode/dispatch stage, dispatch unit **20** decodes and dispatches one or more instructions from instruction queue **19** to the appropriate ones of execution units **22**, **28**, and **30**. Also during the decode/dispatch stage, dispatch unit **20** allocates a rename buffer within GPR rename buffers **33** or FPR rename buffers **37** for each dispatched instruction's result data. According to a preferred embodiment of the present invention, processor **10** dispatches instructions in program order and tracks the program order of the dispatched instructions during out-of-order execution utilizing unique instruction identifiers. In addition to an instruction identifier, each instruction within the execution pipeline of processor **10** has an rA tag and a rB tag, which indicate the sources of the A and B operands for the instruction, and a rD tag that indicates a destination rename buffer within GPR rename buffers **33** or FPR rename buffers **37** for the result data of the instruction.

During the execute stage, execution units **22**, **28**, and **30**, execute instructions received from dispatch unit **20** opportunistically as operands and execution resources for the indicated operations are available. After execution has finished, execution units **22**, **28**, and **30** store result data within either GPR rename buffers **33** or FPR rename buffers **37**, depending upon the instruction type. Then, execution units **22**, **28**, and **30** notify completion unit **40** which instructions have finished execution. Finally, instructions are completed by completion unit **40** in program order by transferring result data from GPR rename buffers **33** and FPR rename buffers **37** to GPRs **32** and FPRs **36**, respectively.

Processor **10** is capable of executing multi-register load/store instructions, which load and store data in a plurality of general purpose registers from and to memory. In particular, in the preferred embodiment having a PowerPC™ microprocessor, microprocessor **10** will execute a load multiple instruction (lmw), which loads multiple words from memory, and a store multiple instruction (stmw), which stores multiple words to memory.

These multi-register instructions are retrieved from instruction cache **14** by sequential fetcher **17** and loaded into instruction queue **19**. Upon dispatch of a multi-register instruction by dispatch unit **20**, LSU **28** will begin execution of the multi-register instruction. Also, upon dispatch of the instruction, a number of registers in GPR **32** identified in the multi-register instruction are allocated to the instruction.

In the preferred embodiment, the load multiple instruction requires that up to 32 contiguous registers be loaded with up to 32 contiguous words from memory.

For example, the instruction "lmw r3, r2, r1" will load registers 3-31 with data found at location <r2 + r1> in memory. Thus, for this example, the first register to be loaded will be register 3 (r3). LSU **28** will then proceed to load register 4, register 5, and so on until all the registers up to and including register 31 have been loaded. At that point the load multiple instruction has finished execution. This is reported to completion unit **40**, which completes the instruction by committing it to architected registers in the system.

Referring to **Figure 2**, there is shown a timing diagram of the cycles required to handle the load multiple instruction and subsequent fixed-point instructions. The load multiple instruction (Load Mult) is fetched (F) from instruction cache **14** by sequential fetcher **17** during cycle **1**. The instruction is decoded (Dec) during cycle **2** and dispatched (Disp) by dispatch unit **20** to LSU **28** during cycle **3**. LSU **28** executes (E) the load multiple instruction during cycled **4-7**, and the instruction is completed (C) by completion unit **40** during cycle **8**.

In this example, four general purpose registers are loaded. In a preferred embodiment, the load multiple instruction would be formatted as lmw r28, r2, r1. This instruction would load registers 28-31, and, as seen in **Figure 2**, one register is loaded per system clock cycle for cycles **4-7**.

In the prior art, any fixed-point instructions subsequent to the load multiple instruction would be serialized so that they would not be fetched until after the load multiple instruction completed. This enables coherency of the operand data utilized by the subsequent fixed-point instructions to be maintained. Therefore, in the example of **Figure 2**, any subsequent fixed-point instructions could not be fetched until cycle 9 in the prior art.

According to the present invention, a deallocation mechanism is provided that allows fixed-point instructions waiting in the instruction buffer to be dispatched prior to completion of the load multiple instruction. As seen in **Figure 2**, prior to cycle **4**, registers 28-31 are allocated to the load multiple instruction. However, once register 28, for example, has been loaded by the load multiple instruction, this resource is deallocated to allow its contents to be used as operand data for subsequent instructions. As a consequence, subsequent fixed-point instructions that are dependent upon the results of the load multiple instruction can be dispatched to other functional units prior to the completion of the load multiple instruction.

Processor **10** maintains a scoreboard or table for all general purpose registers (GPR) **32** that lists each register and indicates when the load multiple instruction has loaded that associated register. Dispatch unit **20** accesses the scoreboard to determine whether subsequent instructions can be dispatched. For example, as seen in **Figure 2**, consider how to handle a load multiple instruction (Load Mult) followed by a first and second fixed-point instruction (FX Inst 1 and FX Inst 2), as represented by the following instruction sequence:

lmw r28, r2, r1
add r2, r2, r28
add r3, r3, r30 (Note: the "adds" are fixed-point instructions to add the contents of a first register to the contents of a second register and store the result operand in the first register)

As can be seen in **Figure 2**, FX Inst 1 can be dispatched as soon as register 28 is released by the load/store unit to the scoreboard. During cycle **4** the load/store unit has executed the load multiple instruction for register 28, so this register will be deallocated on the scoreboard. In the next subsequent cycle, dispatch unit **20** will dispatch FX Inst 1 to FXU **22** because the source operand data for this instruction is now available in the general purpose registers. This instruction is executed by FXU **22** during cycle **6**, but completion unit **40** does not complete the instruction until cycle **9** in order to guarantee coherency of register data. As can be seen from **Figure 2**, FX Inst 2 is fetched and decoded during cycles **3** and **4**. However, dispatch unit **20** does not dispatch this instruction to FXU **22** until after the load multiple instruction has loaded register 30 during cycle **6** and has indicted this load on the scoreboard. Dispatch unit **20** reads the deallocation of register 30 and dispatches the second fixed-point instruction during cycle **7**, and FXU **22** executes the instruction during cycle **8**. Completion unit **40** does not complete this instruction until cycle **10** because all the fixed-point instructions must be completed in programming sequence.

As shown in this example, there has been an improvement of up to eight clock cycles to the dispatch and execution of the subsequent fixed-point instructions (assuming single ported register files and caches). This is a substantial increase in processor efficiency over the prior art. In fact, in certain examples, an improvement of up to 32 clock cycles could be realized. As will be appreciated, this enhanced performance is only limited by the depth of the completion buffer in the completion unit as to how many instructions can be in the execute pipeline before the load-multiple completes.

Early deallocation of resources is also accomplished during the execution of a store multiple instruction. In the preferred embodiment, the store multiple operation requires that up to 32 contiguous registers be stored to up to 32 contiguous word locations in memory. For example, the store multiple instruction stmw r3, r2, r1 will store the contents of register 3 - register 31 to the memory located at <r2 + r1>. According to the present invention, upon dispatch of the store multiple instruction, additional subsequent fixed-point instructions can be dispatched to the other fixed-point execution units in the microprocessor, unconditionally. As before, these instructions must complete in programming sequence, but execution may begin immediately after dispatch of these instructions.

In the present invention, it is recognized that the store multiple instruction does not have to be serialized with subsequent fixed-point instructions. Consequently,

the entire contiguous set of registers required for the store multiple instruction does not have to be allocated exclusively to that instruction, and instead can be used as a source operand resource for subsequent instructions. However, any results of the subsequent instructions must be stored in GPR rename buffers **33** until the store multiple instruction completes to prevent the writing of registers which have not yet been stored. Upon completion of the store multiple instruction, the subsequent instructions can be completed by transferring the result operands from GPR rename buffers **33** to an architected register in GPR **32**.

Referring now to **Figure 3**, there is shown a timing diagram of the cycles in which a store multiple instruction (Store Mult) and two fixed-point instructions (FX Inst3 and FX Inst4) are handled in the microprocessor, in accordance with the preferred embodiment of the present invention. As an example, consider the handling of the following instruction sequence:

```
stmw r28, r2, r1
add r2, r2, r28
add r3, r3, r30
```

As can be seen in **Figure 3**, the store multiple instruction is fetched during cycle **1**, decoded in cycle **2**, dispatched in cycle **3**, executed by the load/store unit **28** during cycles **4-7**, and is completed in cycle **8**. In accordance with the present invention, FX Instl and FX Inst2 are dispatched as soon as possible after the prior store multiple instruction has been dispatched in cycle **3**. FX Inst1 is dispatched during cycle **4**, and because only one instruction may be fetched per cycle, FX Inst2 is dispatched during cycle **5**. These fixed-point instructions may execute immediately, as is done in cycles **5** and **6**, because the operand data required for execution is already present in registers 28 and 30, regardless of the progress of the store multiple instruction execution. In fact, during cycle **6**, LSU **28** will store the data from register 30 out to memory and FXU **22** will add the operand data contained in register 30 to the operand data contained in register 3. The results of the fixed-point instructions 1 and **2** are held in rename buffers **33** until cycles **9** and **10**, respectively, at which time the result operands are stored to registers 2 and 3, respectively. As has been explained, these fixed-point instructions do not complete until after the store multiple instruction has completed in order to maintain resource coherency.

In summary, the present invention addresses the substantial problem of the increased overhead associated with serialized loads and stores. Such serialization of operations requires that the microprocessor's registers be completely empty prior to dispatch of the serialized operation and that those resources remain allocated for that serialized instruction until completion. In accordance with the present invention, the microprocessor performance is substantially increased by not requiring serialization and by allowing additional subsequent instructions to be executed simultaneously with load and store multiple instructions. Depending upon the microprocessor's completion buffer, a significant number of additional instructions may be executed by the microprocessor during the execution of the multi-register instructions. For example, in the preferred embodiment, with a completion buffer depth of five registers, there is a potential of up to four additional instructions completing without a pipeline stall. Because such load and store multiple instructions can take up to 36 cycles to complete, this provides substantial time savings to enhance microprocessor speed and efficiency.

Claims

1. A method of executing multiple instructions in a superscalar microprocessor, including at least one load multiple instruction that loads to more than one register of a plurality of registers, comprising the steps of:

   dispatching a load multiple instruction to a load/store unit, wherein the load/store unit begins execution of a dispatched load multiple instruction, and wherein the load multiple instruction loads data from memory into a plurality of registers;

   maintaining a table that lists each register of the plurality of registers and that indicates when data has been loaded into each register by the executing load multiple instruction; and

   executing an instruction that is dependent upon source operand data loaded by the load multiple instruction into a register of the plurality of registers indicated by the instruction as a source register, prior to the load multiple instruction completing its execution, when the table indicates the source operand data has been loaded into the source register.

2. The method of claim 1, wherein said dependent instruction does not complete until after said load multiple instruction has completed.

3. A method of executing multiple instructions in a superscalar microprocessor, including at least one store multiple instruction that stores data from more than one register of a plurality of registers to memory, comprising the steps of:

   dispatching a store multiple instruction to a load/store unit, whereupon the load/store unit begins executing the store multiple instruction, wherein the store multiple instruction stores data from a plurality of registers to memory; and

   executing a fixed point instruction that is dependent upon data being stored by the store multiple instruction from a register of the plurality of registers indicated by the fixed point in-

struction as a source register, prior to the store multiple instruction completing its execution, but prohibiting the executing fixed point instruction from writing to a register of the plurality of registers prior to the store multiple instruction completing.

4. The method of claim 3, wherein said dependent instruction does not complete until after said store multiple instruction has completed.

5. A superscalar microprocessor (10) having a plurality of execution units (22, 28, 30) for simultaneously executing a plurality of instructions and being connected to a memory, comprising:

a plurality of registers (32, 33, 36, 37) for selectively storing data resulting from execution by the execution units;
a load/store execution unit (28) that executes load and store instructions which load or store data to or from the plurality of registers;
a fixed point execution unit (22) for performing fixed point operations on operand data stored in source registers of the plurality of registers;
and a dispatcher (20) that maintains a table listing at least some of the plurality of registers and indicating when data has been loaded into each register by a load multiple instruction executing in the load/store execution unit, wherein a load multiple instruction loads data from memory into more than one of the plurality of registers, and wherein the dispatcher dispatches instructions to the plurality of execution units, including the load/store execution unit and the fixed point execution unit, and further wherein the dispatcher dispatches a load multiple instruction to the load/store unit, which begins execution of the load multiple instruction and further, prior to the load multiple instruction finishing its execution, dispatches a fixed point instruction which is dependent upon source operand data loaded by the load multiple instruction into a register of the plurality of registers indicated by the instruction as a source register, when the table indicates the source operand data has been loaded into the source register by the execution of the load multiple instruction in the load/store execution unit.

6. The microprocessor of claim 5, wherein said fixed point instruction does not complete until after said load multiple instruction has completed.

7. The microprocessor of claim 5 or 6, wherein the dispatcher dispatches a store multiple instruction to the load/store unit, which begins execution of the store multiple instruction, wherein a store multiple instruction stores data from more than one of the plurality of registers to memory, and further, prior to the store multiple instruction finishing, dispatches a fixed point instruction, which is dependent upon source operand data stored in a register of the plurality of registers that is being stored out to memory by the executing store multiple instruction, to the fixed point execution unit, wherein the fixed point execution unit executes the dispatched fixed point instruction prior to the store multiple instruction finishing, but does not store the result of the executed fixed point instruction to a register of the plurality of registers prior to the store multiple instruction finishing.

8. A superscalar microprocessor (10) having a plurality of execution units (22, 28, 30) for simultaneously executing a plurality of instructions and being connected to a memory, comprising:

a plurality of registers (32, 33, 36, 37)for selectively storing data resulting from execution by the execution units;
a load/store execution unit (28) that executes load and store instructions which load or store data to or from the plurality of registers;
a fixed point execution unit (22) for performing fixed point operations on operand data stored in source registers of the plurality of registers;
a dispatcher (20) that dispatches instructions to the plurality of execution units, including the load/store execution unit and the fixed point execution unit, wherein the dispatcher dispatches a store multiple instruction to the load/store unit, which begins execution of the store multiple instruction, wherein a store multiple instruction stores data from more than one of the plurality of registers to memory, and further, prior to the store multiple instruction finishing, dispatches a fixed point instruction, which is dependent upon source operand data stored in a register of the plurality of registers that is being stored out to memory by the executing store multiple instruction, to the fixed point execution unit, wherein the fixed point execution unit executes the dispatched fixed point instruction prior to the store multiple instruction finishing, but does not store the result of the executed fixed point instruction to a register of the plurality of registers prior to the store multiple instruction finishing.

9. The microprocessor of claim 8, wherein said fixed point instruction does not complete until after said store multiple instruction has completed.

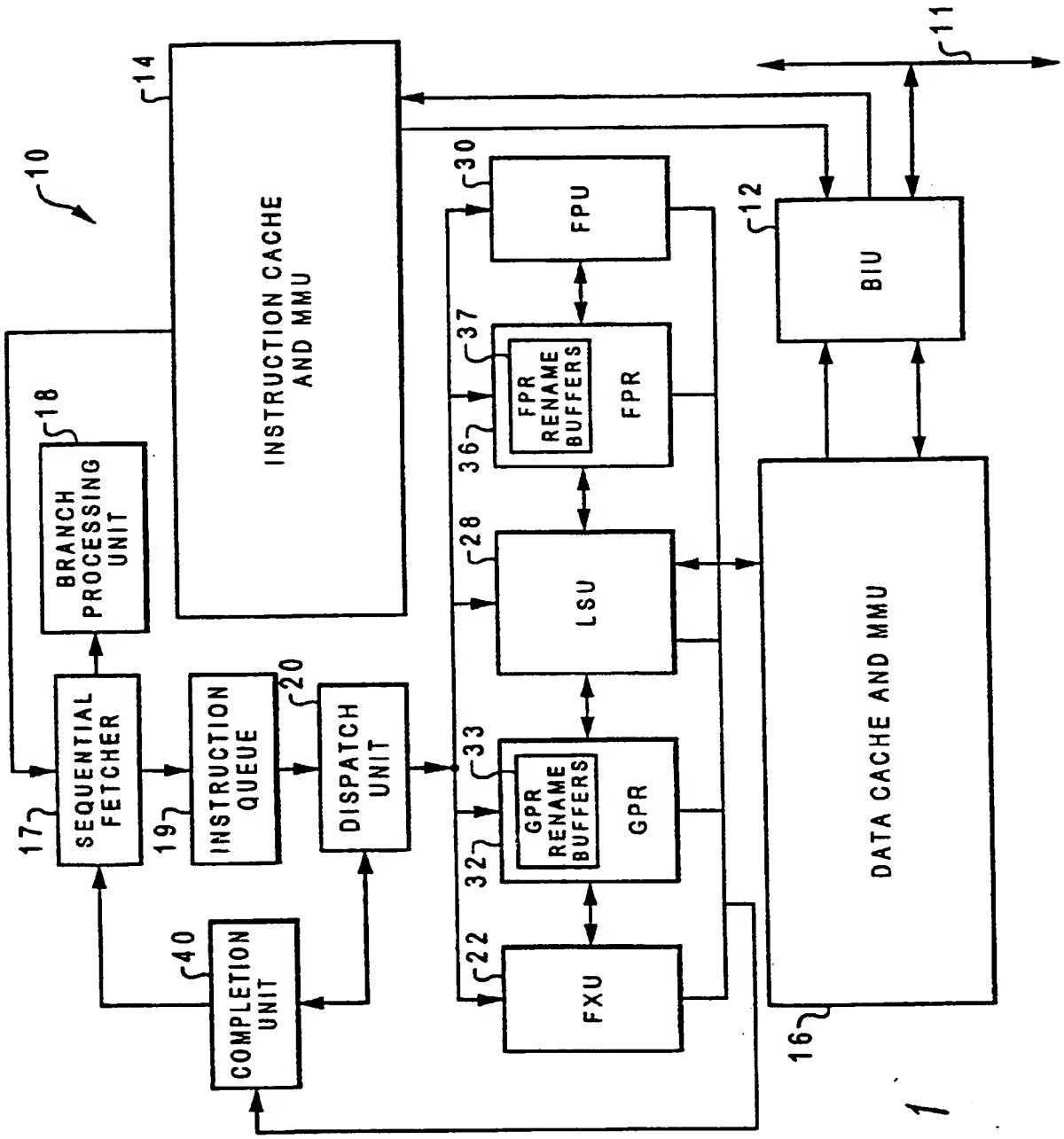

Fig. 1

	1	2	3	4	5	6	7	8	9	10
Load Mult	F	Dec	Disp	E	E	E	E	C		
FX Inst 1		F	Dec		Disp	E			C	
FX Inst 2			F	Dec			Disp	E		C

Fig. 2

	1	2	3	4	5	6	7	8	9	10
Store Mult	F	Dec	Disp	E	E	E	E	C		
FX Inst 1		F	Dec	Disp	E				C	
FX Inst 2			F	Dec	Disp	E				C

Fig. 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 762 270 A3

(12) EUROPEAN PATENT APPLICATION

(88) Date of publication A3:
28.07.2004 Bulletin 2004/31

(43) Date of publication A2:
12.03.1997 Bulletin 1997/11

(21) Application number: **96306249.2**

(22) Date of filing: **29.08.1996**

(51) Int Cl.[7]: **G06F 9/312**, G06F 9/38

(84) Designated Contracting States:
DE FR GB

(30) Priority: **11.09.1995 US 526343**

(71) Applicant: **INTERNATIONAL BUSINESS MACHINES CORPORATION**
Armonk, NY 10504 (US)

(72) Inventors:
- **Kahle, James A.**
  Austin, Texas 78731 (US)
- **Loper, Albert J.**
  Cedar Park, Texas 78613 (US)
- **Mallick, Soummya**
  Austin, Texas 78729 (US)
- **Ogden, Aubrey D.**
  Round Rock, Texas 78681 (US)

(74) Representative: **Davies, Simon Robert**
I B M
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) Microprocessor with load/store operation to/from multiple registers

(57) A load multiple instruction may be executed in a superscaler microprocessor by dispatching a load multiple instruction to a load/store unit, wherein the load/store unit begins execution of a dispatched load multiple instruction, and wherein the load multiple instruction loads data from memory into a plurality of registers. A table is maintained that lists each register of the plurality of registers and that indicates when data has been loaded into each register by the executing load multiple instruction. An instruction is executed that is dependent upon source operand data loaded by the load multiple instruction, prior to the load multiple instruction completing its execution, when the table indicates the source operand data has been loaded into the source register. Also, a store multiple instruction may be executed by dispatching a store multiple instruction to the load/store unit, whereupon the load/store unit begins executing the store multiple instruction, wherein the load/store instruction stores data from a plurality of registers to memory. A fixed point instruction is executed that is dependent upon data being stored by the store multiple instruction prior to the store multiple instruction completing its execution, but the executing fixed point instruction is prohibited from writing to a register of the plurality of registers prior to the store multiple instruction completing.

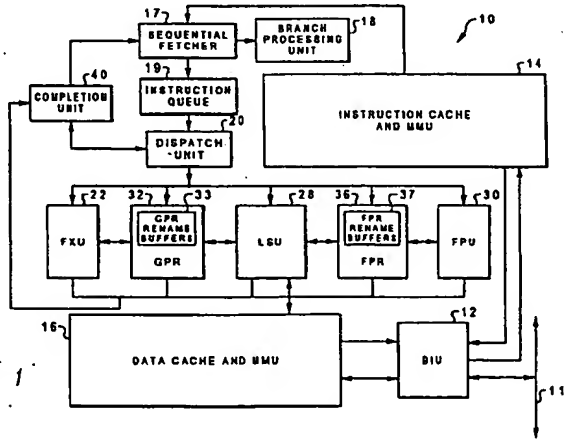

Fig. 1

Printed by Jouve, 75001 PARIS (FR)

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 96 30 6249

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
A	US 5 416 911 A (DINKJIAN ROBERT M ET AL) 16 May 1995 (1995-05-16) * abstract * * column 1, line 1 - column 3, line 52 *	1,2,5-7	G06F9/312 G06F9/38
A	EP 0 651 323 A (ADVANCED MICRO DEVICES INC) 3 May 1995 (1995-05-03) * abstract * * figure 2 *	1,2,5-7	
A	WO 94/08287 A (S MOS SYSTEMS INC) 14 April 1994 (1994-04-14) * abstract * * figure 2 *	1,2,5-7	
			TECHNICAL FIELDS SEARCHED (Int.Cl.6) G06F

~~The present search report has been drawn up for all claims~~

Place of search	Date of completion of the search	Examiner
Munich	16 March 2004	Sadoune, M-M

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)

European Patent Office

Application Number
EP 96 30 6249

CLAIMS INCURRING FEES

The present European patent application comprised at the time of filing more than ten claims.

☐ Only part of the claims have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims and for those claims for which claims fees have been paid, namely claim(s):

☐ No claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims.

LACK OF UNITY OF INVENTION

The Search Division considers that the present European patent application does not comply with the requirements of unity of invention and relates to several inventions or groups of inventions, namely:

see sheet B

☐ All further search fees have been paid within the fixed time limit. The present European search report has been drawn up for all claims.

☐ As all searchable claims could be searched without effort justifying an additional fee, the Search Division did not invite payment of any additional fee.

☐ Only part of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the inventions in respect of which search fees have been paid, namely claims:

☒ None of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the invention first mentioned in the claims, namely claims:

1,2,5-7

European Patent Office

LACK OF UNITY OF INVENTION
SHEET B

Application Number
EP 96 30 6249

The Search Division considers that the present European patent application does not comply with the requirements of unity of invention and relates to several inventions or groups of inventions, namely:

1. claims: 1,2,5-7

   Superscalar processor with "load multiple" instruction using a table of register load indicators to indicate the availability of operands.

   ---

2. claims: 3,4,8,9

   Superscalar processor with "store multiple" instruction and a mechanism to protect the content of registers before they have been written to memory.

ANNEX TO THE EUROPEAN SEARCH REPORT ON EUROPEAN PATENT APPLICATION NO. EP 96 30 6249

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

16-03-2004

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5416911 A	16-05-1995	JP 2620511 B2	18-06-1997
		JP 6242953 A	02-09-1994
EP 0651323 A	03-05-1995	US 5878245 A	02-03-1999
		DE 69433339 D1	24-12-2003
		EP 0651323 A1	03-05-1995
		EP 0952517 A2	27-10-1999
		JP 7182167 A	21-07-1995
		US 6298423 B1	02-10-2001
WO 9408287 A	14-04-1994	DE 69329778 D1	25-01-2001
		DE 69329778 T2	26-04-2001
		EP 0663083 A1	19-07-1995
		HK 1014780 A1	25-01-2002
		JP 8504977 T	28-05-1996
		JP 2000148480 A	30-05-2000
		JP 2000148481 A	30-05-2000
		JP 2000148490 A	30-05-2000
		JP 2000181708 A	30-06-2000
		JP 2000148483 A	30-05-2000
		JP 2000148491 A	30-05-2000
		JP 2000148492 A	30-05-2000
		JP 2000148493 A	30-05-2000
		JP 2000148494 A	30-05-2000
		JP 2003131870 A	09-05-2003
		JP 2003177913 A	27-06-2003
		JP 2003131871 A	09-05-2003
		JP 2003131872 A	09-05-2003
		KR 248903 B1	15-03-2000
		WO 9408287 A1	14-04-1994
		US 2003056089 A1	20-03-2003
		US 2002188829 A1	12-12-2002
		US 5659782 A	19-08-1997
		US 6735685 B1	11-05-2004
		US 6434693 B1	13-08-2002
		US 6230254 B1	08-05-2001
		US 5557763 A	17-09-1996
		US 5987593 A	16-11-1999

EPO FORM P0459

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82